# What Do You Mean my Database Server Core Dumped?

## How to Inspect Postgres Core Dumps

Syed Faisal Akber

VMware, Inc.

2014-05-23

# Introduction

- Quick introduction to core dump analysis
- Requires knowledge of application source code
- Proper analysis can take time
- Knowledge of C is essential
- Knowledge of assembly is recommended
- Sometimes it is better to look at the “environment” instead

# What is a core dump?

# What is a core dump?

- A core dump is a copy of
  - memory contents
  - CPU registers
  - and other information
- stored in a disk file

# Why does a core dump get generated?

- A program did something bad
  - Divide by zero
  - Bus errors (Segmentation violations)
- Someone or something forced it
  - `SIGABRT` (Abort)
  - `SIGSEGV` (Segmentation Violation)
  - `SIGFPE` (Floating-Point Error)
  - `SIGILL` (Illegal Instruction)
- Hardware issues?

See `signal`(7), `kill`(1), and `kill`(2) manual pages for more information.

# Ensuring a core dump is generated

- User limit (`ulimit`) or resource limit (`rlimit`) must be set
  - On *sh* or *bash*, run `ulimit -c unlimited`
  - On *csh*, run `limit -c unlimited`
  - On *tcsh*, run `limit coredumpsize unlimited`
- Have enough space on disk
- Ensure that ulimit is setup in the shell before starting Postgres
- Core dump will be in `$PGDATA`

# Inspecting Core Dumps

- Basic inspection involves usage of a few tools
- Detailed review requires GDB

# Basic Inspection Tools I

- `file`

```
$ file core
core: ELF 64-bit LSB  core file x86-64, version 1 (SYSV), SVR4-style, from
'/opt/postgres/bin/postgres -D /var/postgres'
```

- `readelf`

```
$ readelf -h core
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              CORE (Core file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          0 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         35
```

# Basic Inspection Tools II

```
Size of section headers:           0 (bytes)
Number of section headers:         0
Section header string table index: 0
```

- Other tools such as `objdump` are useful as well

# Core Dump Analysis – Prerequisites I

- GDB (GNU Debugger)
- Source code
- An unstripped binary equivalent to what was in memory

```
$ cd src/backend
$ file postgres
postgres: ELF 64-bit LSB  executable, x86-64, version 1 (SYSV), dynamically
linked (uses shared libs), for GNU/Linux 2.6.24,
BuildID[sha1]=90b4b71f515c653a7ce50f0474741852bf873505, not stripped
postgres:     file format elf64-x86-64
$ objdump -h postgres
Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .interp       0000001c  0000000000400238  0000000000400238  00000238  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
...
 12 .text         002e07ba  00000000004590c0  00000000004590c0  000590c0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
...
 23 .data         0000cbe0  0000000000afc7c0  0000000000afc7c0  004fc7c0  2**5
                  CONTENTS, ALLOC, LOAD, DATA
 24 .bss          0005b068  0000000000b093a0  0000000000b093a0  005093a0  2**5
                  ALLOC
```

# Core Dump Analysis – Prerequisites II

```
...
 26 .debug_aranges 00006790  0000000000000000  0000000000000000  005093c4  2**0
                   CONTENTS, READONLY, DEBUGGING
 27 .debug_info    0088e8e4  0000000000000000  0000000000000000  0050fb54  2**0
                   CONTENTS, READONLY, DEBUGGING
```

# Core Dump Analysis – Invoking GDB

```
$ gdb src/backend/postgres -c /var/postgres/core
GNU gdb (Ubuntu 7.7-0ubuntu3) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./postgres...done.
[New LWP 2368]
Core was generated by `/opt/postgres/bin/postgres -D /var/postgres'.
Program terminated with signal SIGABRT, Aborted.
#0  0x00007f8e6c4fac13 in __select_nocancel ()
    at ../sysdeps/unix/syscall-template.S:81
81      ../sysdeps/unix/syscall-template.S: No such file or directory.
(gdb)
```

# Basic GDB Commands - backtrace

```
(gdb) bt
#0  fib (n=6) at multi0.c:14
#1  0x00000000004006aa in fib (n=7) at multi0.c:16
#2  0x00000000004006aa in fib (n=8) at multi0.c:16
#3  0x00000000004006c5 in fib (n=10) at multi0.c:16
#4  0x00000000004006c5 in fib (n=12) at multi0.c:16
#5  0x00000000004006c5 in fib (n=14) at multi0.c:16
...
#87 0x00000000004006aa in fib (n=110) at multi0.c:16
#88 0x00000000004005ea in main (argc=3, argv=0x7fff84af3bd8) at multi.c:16
```

# Basic GDB Commands - `frame`

```
(gdb) f 0
#0  fib (n=6) at multi0.c:14
14              if ((i == 100) && (n == 0))
(gdb)
...
(gdb) f 88
#88 0x00000000004005ea in main (argc=3, argv=0x7fff84af3bd8) at multi.c:16
16              y = fib(b);
```

# Basic GDB Commands - print

```
(gdb) p x
$13 = 1.5882455415227421e+178
(gdb) p/x x
$14 = 0x8000000000000000
(gdb) p &x
$15 = (double *) 0x7fff84af3ad8
(gdb) printf "a is %d\nb is %d\n", a, b
a is 110
b is 110
```

# Basic GDB Commands - x

```
(gdb) x/xg 0x7fff84af3ae0
0x7fff84af3ae0: 0x405b800000000000
(gdb) x/f 0x7fff84af3ae0
0x7fff84af3ae0: 110
(gdb) x/i 0x00000000004005ea
=> 0x4005ea <main+102>: movsd  %xmm0,-0x20(%rbp)
(gdb) x/i 0x00000000004006aa
   0x4006aa <fib+138>:  movsd  %xmm0,-0x10(%rbp)
```

# Basic GDB Commands - `info registers` I

```
(gdb) info reg rsp
rsp            0x7fff84af3ab0   0x7fff84af3ab0
(gdb) info reg rip
rip            0x4005ea 0x4005ea <main+102>
(gdb) info reg rax
rax            0xb800000000000  3236962232172544
(gdb) info reg
rax            0xb800000000000  3236962232172544
rbx            0x0      0
rcx            0x0      0
rdx            0x405b800000000000       4637440978796412928
rsi            0xfffff00000000  4503595332403200
rdi            0x7fff84af3a50   140735419464272
rbp            0x7fff84af2fc0   0x7fff84af2fc0
rsp            0x7fff84af2fb0   0x7fff84af2fb0
r8             0x0      0
r9             0x0      0
r10            0x2      2
r11            0x7fff84af39a8   140735419464104
r12            0x4004a0 4195488
r13            0x7fff84af3bd0   140735419464656
r14            0x0      0
r15            0x0      0
rip            0x40065e 0x40065e <fib+62>
```

# Basic GDB Commands - `info registers` II

```
eflags         0x203    [ CF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
```

# Basic GDB Commands - disassemble I

```
(gdb) disassemble
Dump of assembler code for function fib:
   0x0000000000400620 <+0>:     push   %rbp
   0x0000000000400621 <+1>:     mov    %rsp,%rbp
   0x0000000000400624 <+4>:     sub    $0x10,%rsp
   0x0000000000400628 <+8>:     movsd  %xmm0,-0x8(%rbp)
   0x000000000040062d <+13>:    xorpd  %xmm0,%xmm0
   0x0000000000400631 <+17>:    ucomisd -0x8(%rbp),%xmm0
   0x0000000000400636 <+22>:    jne    0x400643 <fib+35>
   0x0000000000400638 <+24>:    jp     0x400643 <fib+35>
   0x000000000040063a <+26>:    xorpd  %xmm0,%xmm0
   0x000000000040063e <+30>:    jmpq   0x4006ca <fib+170>
   0x0000000000400643 <+35>:    movsd  0x1cd(%rip),%xmm0        # 0x400818
   0x000000000040064b <+43>:    ucomisd -0x8(%rbp),%xmm0
   0x0000000000400650 <+48>:    jne    0x40065e <fib+62>
   0x0000000000400652 <+50>:    jp     0x40065e <fib+62>
   0x0000000000400654 <+52>:    movsd  0x1bc(%rip),%xmm0        # 0x400818
   0x000000000040065c <+60>:    jmp    0x4006ca <fib+170>
=> 0x000000000040065e <+62>:    movsd  0x2009e2(%rip),%xmm0        # 0x601048 <i>
   0x0000000000400666 <+70>:    ucomisd 0x1b2(%rip),%xmm0        # 0x400820
   0x000000000040066e <+78>:    jne    0x400694 <fib+116>
   0x0000000000400670 <+80>:    jp     0x400694 <fib+116>
   0x0000000000400672 <+82>:    xorpd  %xmm0,%xmm0
   0x0000000000400676 <+86>:    ucomisd -0x8(%rbp),%xmm0
   0x000000000040067b <+91>:    jne    0x400694 <fib+116>
   0x000000000040067d <+93>:    jp     0x400694 <fib+116>
   0x000000000040067f <+95>:    movsd  0x2009c1(%rip),%xmm0        # 0x601048 <i>
   0x0000000000400687 <+103>:   divsd  -0x8(%rbp),%xmm0
```

# Basic GDB Commands - disassemble II

```
   0x000000000040068c <+108>:   movsd  %xmm0,0x2009ac(%rip)        # 0x601040 <k>
   0x0000000000400694 <+116>:   movsd  -0x8(%rbp),%xmm0
   0x0000000000400699 <+121>:   movsd  0x177(%rip),%xmm1        # 0x400818
   0x00000000004006a1 <+129>:   subsd  %xmm1,%xmm0
   0x00000000004006a5 <+133>:   callq  0x400620 <fib>
   0x00000000004006aa <+138>:   movsd  %xmm0,-0x10(%rbp)
   0x00000000004006af <+143>:   movsd  -0x8(%rbp),%xmm0
   0x00000000004006b4 <+148>:   movsd  0x16c(%rip),%xmm1        # 0x400828
   0x00000000004006bc <+156>:   subsd  %xmm1,%xmm0
   0x00000000004006c0 <+160>:   callq  0x400620 <fib>
   0x00000000004006c5 <+165>:   addsd  -0x10(%rbp),%xmm0
   0x00000000004006ca <+170>:   leaveq
   0x00000000004006cb <+171>:   retq
End of assembler dump.
```

# Basic GDB Commands - `list` I

```
(gdb) l
11      //      strcpy(end, argv[1]);
12              a = strtod(argv[1], &g);
13              b = strtod(argv[2], &h);
14
15              x = fact(a);
16              y = fib(b);
17
18              printf("%f! = %f\nfib(%f) = %f\n", a, x, b, y);
19              exit(0);
20      }
(gdb) l -
1       #include <stdio.h>
2       #include <string.h>
3       #include <stdlib.h>
4       #include "multi.h"
5
6       int main(int argc, char *argv[])
7       {
8               double a, b, x, y;
9               char *g, *h;
10
(gdb) l fact
4        *
5        */
6       #include "multi.h"
7
8       double fact(double n)
```

# Basic GDB Commands - `list` II

```
9       {
10              if (n == 0)
11                      return 1;
12              return n * fact(n - 1);
13      }
(gdb) l multi0.c:5
1       /*
2        *
3        * multi0.h - Example for calculating fibonacci numbers.
4        *
5        */
6       #include "multi.h"
7
8       double fib(double n)
9       {
10              if (n == 0)
```

# Analyzing the Collected Information

- Start with `backtrace`
- Observe using `print` and `x`
- Compare inputs and outputs of each `frame`
- Use `list` and `disassemble` to follow the flow
- Note where things don't match up to the source code

# Postgres Core Dumps Specifics I

- Core dump will only contain information from one process *not all*
- Core dump contains shared memory areas

```
(gdb) p AutoVacuumShmem
$3 = (AutoVacuumShmemStruct *) 0x7fd47aee3388
(gdb) p *AutoVacuumShmem
$4 = {av_signal = {0, 0}, av_launcherpid = 1348, av_freeWorkers = {head = {
      prev = 0x7fd47aee33c0, next = 0x7fd47aee3430}}, av_runningWorkers = {
    head = {prev = 0x7fd47aee33a8, next = 0x7fd47aee33a8}},
  av_startingWorker = 0x0}
```

- Global variables will allow easier access to status
- Logs may provide more context

# Case Study I

This is a case study of a Postgres process that was killed intentionally. In this case, it is hypothetically assumed that a client connection is unresponsive.

- Use `ps` to find process

```
$ ps -ef
...
pgcon     2625  1343  0 06:02 ?        00:00:00 postgres: pgcon postgres [local]
...
```

- Kill the process

```
$ kill -6 2625
```

- Start GDB

# Case Study II

```
/var/postgres$ gdb ~/postgresql/src/backend/postgres -c core
GNU gdb (Ubuntu 7.7-0ubuntu3) 7.7
...
Core was generated by ‘postgres: pgcon postgres [local] idle      ’.
Program terminated with signal SIGABRT, Aborted.
#0  0x00007fd47b38821d in __libc_recv (fd=9,
    buf=buf@entry=0xb12560 <PqRecvBuffer>, n=n@entry=8192, flags=-1,
    flags@entry=0) at ../sysdeps/unix/sysv/linux/x86_64/recv.c:29
29      ../sysdeps/unix/sysv/linux/x86_64/recv.c: No such file or directory.
(gdb)
```

- Look at the stacktrace

```
(gdb) bt
#0  0x00007fd47b38821d in __libc_recv (fd=9,
    buf=buf@entry=0xb12560 <PqRecvBuffer>, n=n@entry=8192, flags=-1,
    flags@entry=0) at ../sysdeps/unix/sysv/linux/x86_64/recv.c:29
#1  0x000000000059b583 in recv (__flags=0, __n=8192,
    __buf=0xb12560 <PqRecvBuffer>, __fd=<optimized out>)
    at /usr/include/x86_64-linux-gnu/bits/socket2.h:44
#2  secure_read (port=0xf21360, ptr=0xb12560 <PqRecvBuffer>, len=8192)
    at be-secure.c:304
#3  0x00000000005a2f0b in pq_recvbuf () at pqcomm.c:854
#4  0x00000000005a3a65 in pq_getbyte () at pqcomm.c:895
#5  0x000000000064577a in SocketBackend (inBuf=0x7fffd21495c0)
    at postgres.c:344
#6  ReadCommand (inBuf=0x7fffd21495c0) at postgres.c:492
#7  PostgresMain (argc=<optimized out>, argv=argv@entry=0xf021a8,
    dbname=0xf02058 "postgres", username=<optimized out>) at postgres.c:3958
```

# Case Study III

```
#8  0x000000000045a487 in BackendRun (port=0xf21360) at postmaster.c:3996
#9  BackendStartup (port=0xf21360) at postmaster.c:3685
#10 ServerLoop () at postmaster.c:1586
#11 0x0000000000606df5 in PostmasterMain (argc=argc@entry=3,
    argv=argv@entry=0xf00f40) at postmaster.c:1253
#12 0x000000000045adbb in main (argc=3, argv=0xf00f40) at main.c:206
(gdb)
```

- Inspect a few frames

```
(gdb) f 8
#8  0x000000000045a487 in BackendRun (port=0xf21360) at postmaster.c:3996
3996            PostgresMain(ac, av, port->database_name, port->user_name);
(gdb) p port->database_name
$2 = 0xf02058 "postgres"
(gdb) p port->user_name
$3 = 0xf02040 "pgcon"
(gdb) p *port
$4 = {sock = 9, noblock = 0 '\000', proto = 196608, laddr = {addr = {
      ss_family = 1, __ss_align = 3336130842095267443,
      __ss_padding = "5432", '\000' <repeats 107 times>}, salen = 21},
  raddr = {addr = {ss_family = 1, __ss_align = 0,
      __ss_padding = '\000' <repeats 111 times>}, salen = 2},
  remote_host = 0xf1e770 "[local]", remote_hostname = 0x0,
  remote_hostname_resolv = 0, remote_port = 0xf1e690 "",
  canAcceptConnections = CAC_OK, database_name = 0xf02058 "postgres",
  user_name = 0xf02040 "pgcon", cmdline_options = 0x0, guc_options = 0xf020c8,
  hba = 0xf237f0, md5Salt = "#\230", <incomplete sequence \352\240>,
  SessionStartTime = 453992550897907, default_keepalives_idle = 0,
```

# Case Study IV

```
  default_keepalives_interval = 0, default_keepalives_count = 0,
  keepalives_idle = 0, keepalives_interval = 0, keepalives_count = 0,
  gss = 0x0}
(gdb) p *inBuf
$6 = {data = 0xfafc90 "", len = 0, maxlen = 1024, cursor = 0}
```

# Advanced Topics

Below are some topics to research afterwards

- For Extensions, load "symbols" using `add-symbol-file` in GDB
- Future feature enhancement to generate a core dump containing all of the running threads
- Determine which global variables are useful

# Conclusions

- What a core dump is
- How to ensure a core dump gets generated
- Basic GDB usage
- Tips for understanding output from GDB
- Techniques for Postgres Core Dumps
- Case Study

# Questions?

# References


- *Debugging with GDB* `http://bit.ly/1o56H1z`
- *GDB Reference Card* `http://bit.ly/1l0ZzBz`
- Matloff,Norman and Peter Jay Salzman. 2008. *The Art of Debugging with Gdb, Ddd, and Eclipse.* No Starch Press, San Francisco, CA, USA.
- Momjian, Bruce. *Inside PostgreSQL Shared Memory* `http://momjian.us/main/writings/pgsql/inside_`
- Krosing, Hannu and Kirk Roybal. 2013. *PostgreSQL Server Programming.* Packt Publishing, Birmingham, U.K.

# Contact Information

- E-mail: `fakber@vmware.com`
- IRC: In `#postgresql` on `Freenode` as `sfa`
- Twitter: `@vCoreDump`